All normal end-of-transaction database synchronization steps occur regardless of whether the database has been flushed during the transaction.

EJB Timer Service

The EJB Timer Service uses a database to store persistent information about EJB timers. The EJB Timer Service in GlassFish Server is preconfigured to use an embedded version of the Java DB database.

The EJB Timer Service configuration can store persistent timer information in any database supported by the GlassFish Server for persistence. For a list of the JDBC drivers currently supported by the GlassFish Server, see the *GlassFish Server Open Source Edition 3.1 Release Notes*. For configurations of supported and other drivers, see “Configuration Specifics for JDBC Drivers” in *GlassFish Server Open Source Edition 3.1 Administration Guide*.

The timer service is automatically enabled when you deploy an application or module that uses it. You can verify that the timer service is running by accessing the following URL:

```
http://localhost:8080/ejb-timer-service-app/timer
```

To change the database used by the EJB Timer Service, set the EJB Timer Service’s Timer DataSource setting to a valid JDBC resource. If the EJB Timer Service has already been started in a server instance, you must also create the timer database table. DDL files are located in *as-install*`/lib/install/databases`.

Using the EJB Timer Service is equivalent to interacting with a single JDBC resource manager. If an EJB component or application accesses a database either directly through JDBC or indirectly (for example, through an entity bean’s persistence mechanism), and also interacts with the EJB Timer Service, its data source must be configured with an XA JDBC driver.

You can change the following EJB Timer Service settings. You must restart the server for the changes to take effect.

Minimum Delivery Interval
: Specifies the minimum time in milliseconds before an expiration for a particular timer can occur. This guards against extremely small timer increments that can overload the server. The default is `1000`.

Maximum Redeliveries
: Specifies the maximum number of times the EJB timer service attempts to redeliver a timer expiration after an exception or rollback of a container-managed transaction. The default is `1`.

Redelivery Interval
: Specifies how long in milliseconds the EJB timer service waits after a failed `ejbTimeout` delivery before attempting a redelivery. The default is `5000`.

Timer DataSource
    Specifies the database used by the EJB Timer Service. The default is `jdbc/__TimerPool`.

For information about the `asadmin list-timers` and `asadmin migrate-timers` subcommands, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

You can use the `--keepstate` option of the `asadmin redeploy` command to retain EJB timers between redeployments.

The default for `--keepstate` is false. This option is supported only on the default server instance, named `server`. It is not supported and ignored for any other target.

If any active EJB timer fails to be preserved or restored, *none* of the EJB timers will be available when the redeployment is complete. However, the redeployment continues and a warning is logged.

To preserve active timer data, GlassFish Server serializes the data and saves it in memory. To restore the data, the class loader of the newly redeployed application deserializes the data that was previously saved.

For more information about the `asadmin redeploy` command, see the *GlassFish Server Open Source Edition 3.1 Reference Manual*.

▼ To Deploy an EJB Timer to a Cluster

This procedure explains how to deploy an EJB timer to a cluster. Included in this procedure is an optional step to deploy the default `jdbc/__default` timer that is configured by default in GlassFish Server 3.1. In most production environments, it is recommended that you create your own timer rather than using this default timer.

Before You Begin If creating a new timer resource, the resource should be created before deploying applications that will use the timer. Also note that Step 3 is requited only when the DAS and the clustered instances are on different machines.

1 Start Derby on the DAS machine.

```
asadmin start-database
```

2 If the DAS and the target cluster are running, stop the cluster, stop the domain, and then restart the domain.

```
asadmin stop-cluster cluster-name
asadmin stop-domain domain-name
asadmin start-domain domain-name
```

3 Execute the following command:

```
asadmin set resources.jdbc-connection-pool.DerbyPool.property.serverName=DAS-machine-name
```

4 **Execute the following command:**

```
asadmin create-resource-ref --target cluster-name jdbc/__default
```

5 **Execute the following command:**

```
asadmin set configs.config.cluster_name-config.ejb-container.ejb-timer-service.timer-
datasource=jdbc/__default
```

6 **Start the cluster.**

```
asadmin start-cluster cluster-name
```

7 **(Optional) Restart the Derby database.**

```
asadmin stop-database
asadmin start-database
```

Using Session Beans

This section provides guidelines for creating session beans in the GlassFish Server environment.

The following topics are addressed here:

- "About the Session Bean Containers" on page 151
- "Stateful Session Bean Failover" on page 153
- "Session Bean Restrictions and Optimizations" on page 158

Information on session beans is contained in the Enterprise JavaBeans Specification, v3.1.

About the Session Bean Containers

Like an entity bean, a session bean can access a database through Java Database Connectivity (JDBC) calls. A session bean can also provide transaction settings. These transaction settings and JDBC calls are referenced by the session bean's container, allowing it to participate in transactions managed by the container.

A container managing stateless session beans has a different charter from a container managing stateful session beans.

The following topics are addressed here:

- "Stateless Container" on page 151
- "Stateful Container" on page 152

Stateless Container

The *stateless container* manages stateless session beans, which, by definition, do not carry client-specific states. All session beans (of a particular type) are considered equal.